Available online at www.sciencedirect.com

SciVerse ScienceDirect

Procedia Computer Science 19 (2013) 248 – 255

Procedia Computer Science

The 4th International Conference on Ambient Systems, Networks and Technologies (ANT 2013)

BLOOM FILTER SUPPORTING DISTRIBUTED POLICY-BASED MANAGEMENT IN WIRELESS SENSOR NETWORKS

Nidal Qwasmi, Ramiro Liscano

University of Ontario Institute of Technology
2000 Simcoe Street North
Oshawa, Ontario, Canada, L1H 7K4
Nidal.Qwasmi@uoit.ca, Ramiro.Liscano@uoit.ca

Abstract

Wireless Sensor Networks (WSN) are particularly special due to many characteristics, such as a working environment that makes maintenance and support a challenge; and hardware resources, particularly memory, processing and battery power, that make it capable of handling only limited software. Consequently, the administration of WSN is becoming a challenge. To overcome these limitations we proposed Distributed Policy-Based Management (DBPM) framework. Our proposed framework is expected to conceal the complexity of administrating policies operations from the users by simplifying the deployment processes. Bloom Filter is an elegant data structure that answers the membership inquiry with no false negative and manageable false positive. In this paper we propose utilizing Bloom Filter in Distributed Policy-Based Management (DPBM) environment in WSN to confirm the existence of any policy within the WSN, which will reduce the traffic within the network as well as preserve the sensor node energy.

KeyWords: Bloom Filter; Distributed Policies; hashing algorithms; policy-based management; Internet of Things.

1. Introduction

Distributed mechanisms have been used in wide knowledge areas to resolve resource constraints such as distributed computing, distributed file system, distributed learning, and distributed manufacturing. In this paper we apply the same concept of sharing other network node resources to achieve our proposed framework [2] objectives. Due to hardware resource limitations in WSN, devices on WSN may hold up to 20 policies at any given time [3]. This number may not be sufficient at all times; therefore, a dynamic deployment of the policies is necessary to utilize the node resources and accurately execute the required policies. These limitations have a huge impact on restricting the management capabilities and number of tasks that can be performed on the device and on the WSN as a whole. The architecture of many existing and proposed policy-based WSN platforms relies on a

Selection and peer-review under responsibility of Elhadi M. Shakshuki
doi:10.1016/j.procs.2013.06.036

local policy repository on the node to access any required policy [3] [4] [5]. This architecture choice raises many issues, mainly exposing the users to serious difficulties since they have to store policies on the targeted node only, creating serious administrative difficulties. Due to the nature of limited resources on the sensor node memory as discussed by Zhu et al. [3], it is quite possible for a policy-based WSN network to have more policies than the sensor node capacity. Such cases may exist as a result of the remarkable growth in The Internet of Thing (IoT) which is expected to have more than 50 billion objects in 2020. The number of policies in the WSN is directly connected to the number of constraints that can be created on the WSN, which logically equal the number of functions that can be performed on the WSN. Therefore, the more policies the WSN can accommodate the more management functions (constraints) that users of the WSN can perform.

A policy, by its basic definition, is a constraint on the system behaviours, which can also be expressed using natural languages or mathematical notation. However, neither of these two extremes is an ideal approach to be used by computer systems [23]. Policy-based systems try to strike a balance between the two extremes by creating a policy language that can fulfil the requirements of the targeted system. Hence, policy languages are declarative and not procedural, which means they express the constraints on the system behaviours but do not specify how these constraints ought to be enforced. Policy structure may be varied depending on the system and application requirements, but the most important part of any policy is the policy key (ID). The policy key plays a very crucial role in any system because it will be used throughout the whole network to locate the targeted policy. For that reason this paper propose to utilize Bloom Filter to inquire about the existence of any policy within the network before wasting sensor node energy on looking up a policy that may not be exist. Bloom Filter is an elegant data structure that answers the membership inquiry with no false negative (the check result shows that member doesn't not exist "negative" while it is exist) and acceptable false positive (the check result shows that member exists "positive" while it is not exist).

The rest of the paper is structured as follows. In section 2 related works are presented. Thereafter, in section 3 Bloom Filter analysis is been discussed. Hashing algorithms are discussed in section 4, and then followed by Experimental Results in section 5. Finally, Conclusion is given in section 6.

2. Related Work

Burton H. Bloom [6] introduced for the first time the concept of using a hashing function technique to trade-off between space and time with some allowable error. This technique later named Bloom Filter which is an elegant data structure that answers the membership inquiry with no false negative and acceptable false positive.

Adam Kirsch et al. [9] discussed the benefits of using fewer hashing functions to build Bloom Filter array. In this research authors mathematically proofed that only two hashing functions are necessary to use the Bloom Filter array without any loss in the asymptotic false positive probability. The proposed method is using two hashing functions $h_1(x)$ and $h_2(x)$ to generate k number of new hashing functions in the form of $g_i(x) = h_1(x) + ih_2(x)$ where i between 0 and k-1.

Thanh et al. [17] in their survey listed different hashing algorithms and compared them in term of energy efficient, scalability and data storage/lookup efficiency. Thanh et al. cover many hashing algorithms such as Geographic hash table (GHT) [18], Chord for sensor networks (CSN) [19], Virtual Ring Routing (VRR) [20], Topology based Distributed Hash Table (T-DHT) [21], Cell Hash Routing (CHR) [22], and ScatterPastry [7].

Prosenjit Bose et al. [10] study the false positive rate in Bloom Filter analysis provided by Burton H. Bloom [6]. Authors claim that Bloom's analysis is inaccurate because it underestimates the false-positive rate and they provided a new analysis, but we believe that the rate difference between the two analyses is negligible and apply only to specific cases.

This research propose a new implementation of Bloom Filter in the Distributed Policy-Based Management (DPBM) in WSN domain to confirm the existence of any policy within the WSN, which will reduce the traffic within the network as well as preserve the sensor node energy.

3. Bloom Filter Analysis

Bloom filter was first developed in 1970 by Burton H. Bloom [6]. It is a compact data structure used for making a decision on item membership to a set of data items. Consider a set $S = \{s_1, s_2, \cdots, s_n\}$ of n members, an array $A = \{a_1, a_2, \cdots, a_m\}$ of m members (bits) with initial value of zero for all members (bits). H is a set of independent hash functions $H = \{h_1, h_2, \cdots, h_k\}$ each with output range between 1 and m. For optimal result k has to be according to the formula $k = \ln 2^{(m/n)}$. To add a member to the set S, each bit at positions $h_1(a), h_2(a), \ldots, h_k(a)$ in array A is set to 1. Any bit may be set to 1 many times. To check for membership on item $g \in S$, all bits at positions $h_1(b), h_2(b), \ldots, h_k(b)$in array A has to be set to 1 even though there is still a probability that our conclusion is wrong (called false positive) but we can control the probability of the false positive by selecting the optimal number of hash function as well as the size of the Bloom Filter array. Thus, it is certainly that $g \notin S$ if any bit of them is set to zero.

Bloom Filter sounds a very effective algorithm; however it raises many questions such as the following:

What is the optimal filter array size?
What is the membership test performance?
What is the acceptable limit of false positive rate?
What is the trade-off in servicing the filter?
What is the acceptable trade-off between the actual member lookup test and membership test?

To decide on the optimal filter size, let's assume n keys has been added to the filter F with size m (bits) using a k number of hash functions. Then the probability that a particular bit is still has the value of zero is $(1 - \frac{1}{m})^{kn}$. The probability of a false positive in this case is in (1).

$$\left(1 - \left(1 - \frac{1}{m}\right)^{kn}\right)^k \approx (1 - e^{kn/m})^k \tag{1}$$

Although there is other research [10] claims this equation inaccurate and underestimates the false-positive rate but we believe that the rate difference between the two analyses is negligible and apply only to specific cases. The previous formula can be simplified to (2)

$$k = \ln 2^{(m/n)} \tag{2}$$

We can infer that the optimal number of hash functions is $k = \ln 2^{(m/n)}$. We can also infer that the filter size m (bits) can be obtain by (3)

$$m = \frac{n\,k}{\ln 2} \tag{3}$$

There are some implementations for Bloom Filter in Wireless Sensor Network such as using Bloom filters in Content-based routing [8]. In addition, it has many other implementations in databases, computer networks, social networks, and cryptography.

4. Hashing algorithms

Due to hardware resource limitations, hashing algorithms in the sensor node need to be lightweight, independent, uniformly distributed, and as fast as possible. In this research we adapt the proposed method in [9] which is based on selecting two hashing functions $h_1(x)$ and $h_2(x)$ as a base to generate *k* more new hashing functions in the form of (4) Where *i* between 0 and *k-1*.

$$g_i(x) = h_1(x) + ih_2(x) \quad (4)$$

Moreover the Experimental Results section (section 5) shows that the intersections of the false-positive probability curve with the optimal hashing function line is between 1 and 2 for the two test sample size of 1,024 and 18,000 members.

There are many hashing function that we can choose from. However our criteria are to have a hashing function that is lightweight, independent, uniformly distributed, and as fast as possible. For that purpose we believe that potential hashing functions can be shortlisted as the following:

- *Additive hash*: the simplest hashing algorithm with weak performance. The algorithm is to add the value of the characters in a string together.
- *XOR hash*: simple algorithm with less than average performance. The algorithm is to XOR the value of the characters in a string together.
- *Rotating hash*: similar to XOR hash but with multiple XOR operations. This algorithm would consider the minimal acceptable hashing algorithm.
- *Bernstein hash*: an algorithm created by Dan Bernstein. The algorithm adding the characters of a string and multiply the result with a constant value of 33. The performance results were not great which led to the creation of modified algorithm called Modify Bernstein. The new algorithm was the same except it replace the addition operation with XOR.
- *Shift-Add-XOR hash*: very efficient algorithm for all type of data. It is similar to Rotating hash except it replaces the multiplication with addition and chooses different constant number for rotation. More detail information about this algorithm can be found in [13].
- *One-at-a-Time hash*: created by Bob Jenkins. It performs very well and it consists of multiple shift, addition and XOR operations.
- *FNV series*: it referred to its author names Glenn Fowler, Landon Curt Noll and Phong Vo. FNV algorithm is a series of XORs and multiplications.

Table 1 shows a comparison of some hashing algorithms. The size-1000 column represents the smallest hash table size garter than 1,000. Collision column represents the number of collisions occurred when hashing 38,470 English words to 32bit value. Based on the result in Table 1 and the review of the algorithm we choose to use one-at-a-time and Shift-Add-XOR (similar to rotating algorithm with better performance) hashing algorithms for our experiments.

Table 1 Hashing algorithms comparison[12][14]

Name	size-1000	Speed	Collision
Additive	1,009	5n+3	37,006
Rotating	1,009	6n+3	24
One-at-a-Time	1,024	9*n* +9	0
Bernstein	1,024	7*n* +3	4
Pearson	1,024	12n+5	0
CRC	1,024	9n+3	1
Generalized	1,024	9n+3	0
Universal	1,024	52n+3	0
Zobrist	1,024	10n+3	1
MD4	1,024	9.5n+230	1

5. Performance Evaluation

It was necessary before we start our experiment to define some necessary environment parameters which are Bloom Filter size, member's sample size, number of hashing functions, and hashing algorithm. To find the reasonable Bloom Filter size and number of hashing functions, we rely on the Bloom Filter analysis formulas provided in section 3. Also we consider using the proposed method in [9] to create more hashing functions to enhance the false positive probability.

To decide on member's sample size we assume that a reasonable member's sample size is 1,024 members (policy) based on the fact that conventional policy platform can accommodate up to 20 policies on each node. Therefore the 1,024 members (policy) divided on 20 policies for each node will give us around 51 nodes. That is considered a reasonable wireless sensor network size. On the other end of the spectrum we analyzed the assumption of having 18,000 members (policy) which translate to 900 nodes (18,000/20=900) which considered the largest single wireless sensor network have been implemented.

Figure 1 shows the analysis of 1,024 members sample size. It shows that the intersection between the false positive curve and hashing functions number line lie between the value of 1 and 2 (hashing functions) which have a false positive probability between 0.2 and 0.4. We can also infer from the graph that the optimal Bloom Filter array size is a round 300 byte.

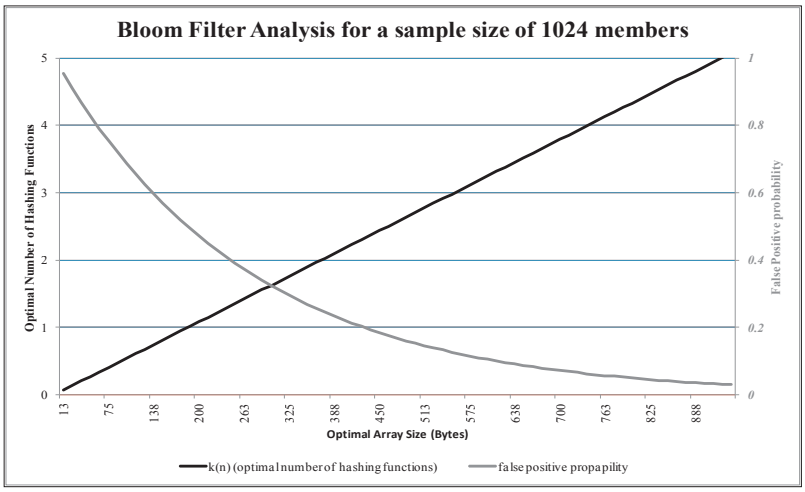

Figure 1 Bloom Filter Analysis for a sample of 1,024 members

Figure 2 shows the analysis of 18,000 members sample size. It shows that the intersection between the false positive and hashing functions number lines lie between the value of 1 and 2 (hashing functions) which have a false positive probability between 0.2 and 0.4. We can also infer from the graph that the optimal Bloom Filter array size is a round 5,225 byte.

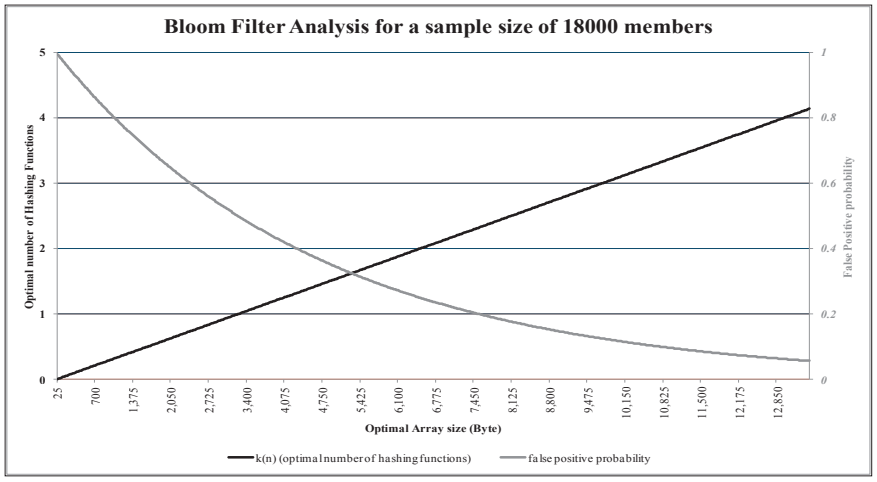

Figure 2 Bloom Filter Analysis for a sample of 18,000 members

For our experiment simulation we used Tinyos-NesC [15] to code the hashing algorithm on MicaZ platform and Avrora simulation software [16] to simulate the experiment. We assumed that the policy ID consists of 32 characters (”0123456789abcdefghijklmnopqrstuvwxyz”). The experiment results for 1,024 members are shown in Table 2.

Table 2 Experiment result for 1,024 members

Hashing Algorithm Name	Time (μs)	Cycle	μJ/Cycle	Energy Consumption (μJ)	Members	Total Time μs	Energy Consumption (μJ)
One_At_a Time	51	176	0.0031	0.5419	1,024	52,224	554.8913
SAX	75	165	0.0031	0.5080	1,024	76,800	520.2106
Total	126	341	0.0031	1.0499	1,024	129,024	1,075.1020

The results for 18,000 members are shown in Table 3. The result values in both tables include running the hashing algorithm and updating the Bloom Filter. The two experiments result tables clearly show the amount of resources the Bloom Filter would need from a sensor node which is very insignificant. For each lookup or update transaction, the sensor node will spend 126μs and used 1.05μJ of energy. In the first case of 1,024 members (policy) the total time needed is 129,024μs, and total energy of 1,075.1μJ.

Table 3 Experiment result for 18,000 members

Hashing Algorithm Name	Time (μs)	Cycle	μJ/Cycle	Energy Consumption (μJ)	Members	Total Time μs	Energy Consumption (μJ)
One_At_a Time	51	176	0.0031	0.5419	18,000	918,000	9,753.9492
SAX	75	165	0.0031	0.5080	18,000	1,350,000	9,144.3274
Total	126	341	0.0031	1.0499	18,000	2,268,000	18,898.2766

In the second case where 18,000 members (policy) are needed, the total time was 2,268,000μs, and total energy of 18,898.285μJ

6. Conclusion

Bloom Filter has been widely used in many domains, especially databases management systems. In this paper, we show how Bloom Filter can assist Policy-based Management framework for Wireless Sensor Network to inspect the existence of a policy within the WSN with little computation time, minimal energy, and limited traffic. We predict that Policy-based Management framework would play a major role in promoting the Internet of Things (IoT) technology.

As shown in section 5, each lookup or update transaction in Bloom Filter spends 126 µs and consumes 1.0499µJ. If we know that, each AA Alkaline Long-life battery contains 9,360J and each node may have two batteries then we can hypothetically execute (9,360*2/1.0499mJ) ≈ 18 billion transactions. These numbers shows the overhead that Bloom Filter transactions would add to any sensor node, which are insignificant.

References

[1] ZHANG, Wenbo and XU, Haifeng. A Policy Based Wireless Sensor Network Management Architecture. Third International Conference on Intelligent Networks and Intelligent System, 2010. pp. 552-555.

[2] Nidal Qwasmi, Ramiro Liscano, Framework for Distributed Policy-Based Management in Wireless Sensor Networks to Support Autonomic Behavior, The 3rd International Conference on Ambient Systems, Networks and Technologies (ANT-2012), Procedia Computer Science, Volume 10, 2012, Pages 232-239

[3] Yanmin Zhu, Sye Loong Keoh, Morris Sloman, Emil Lupu, Yu Zhang, Naranker Dulay, Finger: An Efficient Policy System for Body Sensor Networks, s.l.: In the Proceedings of the 5th International Conference on Mobile Ad-hoc and Sensor Systems (MASS), Atlanta, Georgia, September 29 - October 2., 2008.

[4] Bourdenas, Themistoklis, Sloman, Morris and C. Lupu, Emil, Self-healing for Pervasive Computing Systems, s.l.: In Proceedings of Workshop on Architecting Dependable Systems (WADS), 2009. pp. 1-25.

[5] Yanmin Zhu, Sye Loong Keoh, Morris Sloman, Emil Lupu, Naranker Dulay, An Efficient Policy System for Body Sensor Networks, s.l. : In the Proceedings of the 14th IEEE International Conference on Parallel and Distributed System, 2008.

[6] B. H. Bloom, "Space/time trade-offs in hash coding with allowable errors," in Proc. Communications of the ACM, vol. 13, July 1970, pp. 422–426.

[7] Al-Mamou, Abd Al-Basset and Labiod, Houda. ScatterPastry: An Overlay Routing Using a DHT over Wireless Sensor Networks, s.l. : IEEE, 2007. 0-7695-3006-0/0.

[8] Riihij, Janne, Jardak, Christine and Mahonen, Petri. Analyzing the Optimal Use of Bloom Filters in Wireless Sensor Networks Storing Replicas. s.l. : IEEE Communications Society subject matter experts for publication in the WCNC 2009 proceedings, 2009.

[9] Adam Kirsch and Michael Mitzenmacher. 2008. Less hashing, same performance: Building a better Bloom filter. Random Struct. Algorithms 33, 2 (September 2008), 187-218. DOI=10.1002/rsa.v33:2 http://dx.doi.org/10.1002/rsa.v33:2

[10] Prosenjit Bose, Hua Guo, Evangelos Kranakis, Anil Maheshwari, Pat Morin, Jason Morrison, Michiel Smid, and Yihui Tang. 2008. On the false-positive rate of Bloom filters. Inf. Process. Lett. 108, 4 (October 2008), 210-213.

[11] Glenn Fowler, Kiem-Phong Vo, Donald Eastlake, The FNV Non-Cryptographic Hash Algorithm, September 25, 2012, http://tools.ietf.org/pdf/draft-eastlake-fnv-03.pdf

[12] Bob Jenkins, September 01, 1997, Dr. Dobb's Journal, Internet: http://www.drdobbs.com/database/algorithm-alley/184410284?pgno=5, September 01, 1997 [Dec. 8, 2012]

[13] M.V. Ramakrishna and Justin Zobel, "Performance in practice of string hashing functions", Proc. of the Fifth International Conference on Database Systems for Advanced Applications, 1997, pages 215−223

[14] Bob Jenkins, Bob Jenkins' Web Site, Internet: http://burtleburtle.net/bob/hash/doobs.html, [Dec. 8, 2012]

[15] TinyOS Website, Internet: http://www.tinyos.net/, [Dec. 8, 2012]

[16] Avrora-the AVR simulation and analysis framework Website, Internet: http://compilers.cs.ucla.edu/avrora/index.html, [Dec. 8, 2012]

[17] Thanh, Vinh Vu, et al. A survey of routing using DHTs over Wireless Sensor Networks, s.l. : The 6th International Conference on Information Technology and Applications (ICITA), 2009. pp. 239-244.

[18] Ratnasamy, Sylvia, et al. Datacentric Storage in Sensornets with GHT, Geographic Hash Table, s.l. : Kluwer Academic Publishers, Mobile Networks and Applications 8, 2003. pp. 427-442.

[19] Ali, Muneeb and Uzmi, Zartash Afzal, CSN: A Network Protocol for Serving Dynamic Queries in Large-Scale Wireless Sensor Networks, Fredericton, N.B, Canada : 2nd CNSR 2004, May 2004. pp. 165–174.

[20] Caesar, M., et al., Virtual Ring Routing: Network routing inspired by DHTs, Pisa, Italy : Sigcomm, September 2006.

[21] Landsiedel, Olaf, Lehmann, Katharina Anna and Wehrle, Klaus, T-DHT:Topology-Based Distributed Hash Tables, s.l. : Proceedings of Fifth International IEEE Conference on Peer-to-Peer-Computing, 2005. pp 143-144.

[22] Araujo, F., et al. CHR: adistributed hash table for wireless ad hoc networks, 6-10, June 2005 : Distributed Computing Systems Workshops, 25th IEEE International Conference.

[23] Agrawal, Dakshi, et al. Policy Technology for self-managing systems. s.l. : IBM Press, October 10, 2008. ISBN: 0-13-221307-9.